Conociendo tuplas en Python

Yan Orestes
20/10/2020

Después de todo, ¿cuál es la diferencia entre tuplas y listas en el lenguaje **Python**? ¡Vea cuándo usar uno y otro, ¡y aclare esta gran duda :)!

Estoy trabajando en una aplicación de mapeo que se basa en coordenadas geográficas para encontrar direcciones. Pero, ¿cómo puedo almacenar estas coordenadas?

Una idea es tener una variable para la latitud y otra para la longitud, más o menos así:

```
latitud = -23.588254
longitud = -46.632477
```

Hasta ahora, bien. Pero quiero trabajar con rutas y distancias, así que necesito más de una coordenada. Entonces, ¿Cómo lo hacemos? ¿Tenemos que crear otras variables de latitud y longitud?

```
latitud = -23.588254
longitud = -46.632477

otra_latitud = 48.8583698
otra_longitud = 2.2944833
```

Bueno ... esto ya se está volviendo extraño, ¿verdad? A medida que aumenta el número de coordenadas, nuestro código se vuelve más confuso y más propenso a errores.

Lo ideal sería no separar **latitud** y **longitud**, y sí dejarlas juntas en una variable, realmente organizadas como una coordenada. Cuando queremos guardar juntos más de un valor en Python, lo primero que nos viene a la mente son ...¡las listas! Usemoslas entonces:

```
alura_coordenadas = [-23.588254, -46.632477]
torre_eiffel_coordenadas = [48.8583698, 2.2944833]
```

Ahora ha mejorado mucho, ¡tenemos nuestras dos coordenadas separadas y organizadas! Pero, ¿una lista es el mejor tipo para lo que queremos hacer en Python?

Los problemas de la lista

Las listas en Python son increíblemente poderosas y útiles, pero ¿son lo ideal para nosotros? En primer lugar, podemos pensar en lo que puede resultar problemático en la práctica, como la mutabilidad de las listas.

Ya sabemos que las listas son mutables, es decir, que se pueden cambiar los valores del mismo objeto lista. Entonces, qué pasaría si alguien agregara un valor en `alura_coordinadas` con el método **append()** o incluso quitara uno con el **remove()**. ¡Esta variable ya no tendría sentido!

Sería peor si alguien simplemente cambiara uno de los valores, haciendo esto, por ejemplo:

```
alura_coordenadas = [-23.588254, -46.632477]
alura_coordinadas [1] = -5,0
```

`alura_coordinadas` ahora señalaría una coordenada ajena a lo que era tener la dirección de Alura São Paulo. En realidad, ¡ahora apunta al medio del Océano Atlántico! No tiene ningún sentido para nosotros ...

Además de esta cuestión práctica, tenemos un grave problema de **semántica**. Usamos una lista para representar las coordenadas, ¿verdad? Pero si es una lista, ¿**es una lista de qué**? Después de todo, son dos valores diferentes – **latitud** y **longitud**.

La propia documentación de Python indica que generalmente, los elementos de una lista son **homogéneos**. Es decir, tienen el mismo tipo y significado. Con nuestras coordenadas

tenemos valores **heterogéneos**, porque representan dos cosas diferentes (para luego formar una sola).

No solo; directamente de una definición de lista en el diccionario: una relación **ordenada** de cosas **relacionadas**. Tenga en cuenta que lo que queremos no es **orden**, pero **estructura**, la relación no es qué latitud viene primero que longitud, sino qué latitud está en la primera posición, mientras que la longitud está en la segunda.

¿Qué, además de la lista, podría usarse para satisfacer mejor nuestra necesidad?

Conociendo las tuplas

Python nos proporciona un tipo que satisface mucho mejor las características que estábamos buscando: las tuplas. Podemos considerar las tuplas similares a las listas, pero sus diferencias son cruciales.

En cuanto a la sintaxis, la tupla se diferencia al reemplazar los corchetes (`[]`) de las listas por paréntesis (`()`):

```
alura_coordenadas = (-23.588254, -46.632477)
print(type(alura_coordenadas))
```

Solo para verificar el tipo:

```
<class 'tupla'>
```

Primero, la diferencia más explícita según la propia comunidad Python es que, las **tuplas son inmutables**. Esto significa que no podemos modificar un mismo objeto tupla, es decir, cambiar una de sus referencias internas (sus valores), ni agregar o eliminar ningún elemento:

```
>>> alura_coordenadas = (-23.588254, -46.632477)
>>>
>>> alura_coordenadas.append(3)
Traceback (most recent call last):
    File "<stdin>",line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

```
>>> alura_coordenadas.remove(-46.632477)
Traceback (most recent call last):
    File "<stdin>",line 1, in <module>
AttributeError: 'tuple' object has no attribute 'remove'
>>>
>>> alura_coordenadas[1] = -5.0
Traceback (most recent call last):
    File "<stdin>",line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
>>> alura_coordenadas
(-23.588254, -46.632477)
```

Aún así, podemos crear un nuevo objeto tupla con el operador suma (+):

```
>>> alura_coordenadas = (-23.588254, -46.632477)
>>>
>>> id(alura_coordenadas)
140197171864136
>>>
>>> alura_coordenadas += (-5,)
>>> alura_coordenadas
(-23.588254, -46.632477, -5)
>>> id(alura_coordenadas)
140197171723936
```

Para una tupla de valor único, siempre debemos poner una coma (,) al final del valor, incluso con los paréntesis, o Python no lo interpretará como tupla.

Otro punto importante y que puede generar confusión con respecto a esta inmutabilidad es que **tuplas pueden contener objetos mutables**, como listas:

```
>>> mi_tupla = ([1,2,3],['a','b','c'])
>>> mi_tupla
([1,2,3],['a','b','c'])
>>>
```

```
>>> id(mi_tupla)
140197171822408
>>>
>>> mi_tupla[0].append(4)
>>> mi_tupla
([1,2,3,4],['a','b','c'])
>>>
>>> id(mi_tupla)
140197171822408
```

"Y cómo, ¡modificamos un valor de nuestra tupla sin cambiar de objeto! ¿Esto significa que no es inmutable?"

No, porque en realidad no modificamos nada en la tupla, ¡ya que las referencias que conserva son las mismas! Modificamos la lista dentro de ella, nuevamente **sin cambiar el identificador**, y así fue permitido.

Además, ¿cuál es la diferencia entre tuplas y listas? Para muchos desarrolladores, aparentemente termina ahí, como si las tuplas fueran **solamente** listas constantes, inmutables. Sin embargo, esto no es del todo cierto.

¿Cuál es la diferencia entre listas y tuplas?

De hecho, técnicamente las diferencias más claras son estas: las tuplas se comportan como listas estáticas. Por lo tanto, aún podemos deducir (y confirmar) que, debido a esto, las tuplas ocupan menos espacio en la memoria en comparación con las listas:

```
>>> ().__sizeof__()
24
>>> ().__sizeof__()
40
```

Pero, ¿y más allá de eso?

Cuando tocamos los problemas de las listas, hablamos de la **semántica**, es decir, lo que hace que nuestro código tenga sentido. Descubrimos que una lista no era realmente ideal

en nuestro caso, como especificaba la propia documentación de Python.

Asimismo, la documentación todavía nos da un sentido para las tuplas, lo que indica que suelen contener una secuencia **heterogénea** de elementos, es decir, elementos de diferentes tipos y significados.

Como ya hemos visto, las listas, por otra parte, son más adecuadas para almacenar valores del mismo significado, es decir, **homogéneo**. Esto no significa que una lista no pueda almacenar valores de diferentes tipos, sino que este es comúnmente el sentido que se le da.

De todos modos, las listas suelen funcionar con **orden**, mientras que las tuplas generalmente funcionan con **estructura**. El primer elemento de nuestra tupla `alura_coordinadas` es la latitud, y el segundo es la longitud: la posición no se puede invertir, porque dan sentido a los valores.

Pensando que las tuplas funcionan con estructura, ¿qué pasaría si pudiéramos especificar qué significa cada valor? Es decir, darle un nombre a cada puesto. ¿Podemos hacer eso?

Crear tuplas nombradas con la función `namedtuple()`

Dado que las tuplas se basan en la estructura de nuestros datos, sería interesante poder nombrarlas. Pensando en ello, en Python, ya existe una forma de implementar nuestras tuplas con campos nombrados, con la función namedtuple(), ubicada en el módulo **collections** de la biblioteca estándar del lenguaje.

La función `namedtuple()` en realidad, es una fábrica de tuplas nombradas. Es decir, nos devuelve un objeto de una subclase de tupla que, además de dar acceso a los elementos por el índice, nos permite dar significado nombrado a cada uno de ellos y acceder a ellos de esta forma:

```
>>> from collections import namedtuple
>>>
>>> Coordenadas = namedtuple('Coordenadas', ['latitud', 'longitud')
>>> alura_coordenadas = Coordenadas (-23.588254, longitud = -46.632477)
>>>
>>> alura_coordenadas
Coordenadas (latitud=-23.588254, longitud=-46.632477)
>>> alura_coordenadas[0]
```

```
-23.588254
>>> alura_coordenadas.latitud
-23.588254
>>> alura_coordenadas[1]
-46.632477
>>> alura_coordenadas.longitud
-46.632477
```

Así ¡tenemos la tupla con más beneficios de legibilidad! Una alternativa a las tuplas nombradas podría ser usar **diccionarios**, que son otro tipo nativo de Python que nos permite mapear objetos, como puedes ver en la documentación del lenguaje;).

Conclusión

Las listas y tuplas son algunas estructuras de datos que se utilizan a menudo al programar. Estas dos estructuras nos permiten guardar nuestros datos agrupados, sin embargo, sus similitudes terminan ahí, como pudimos ver en esta publicación.

Logramos desmitificar la idea de que la tupla es **solamente** una lista inmutable, y entendemos cómo las listas tienen una función semántica diferente a las tuplas.

Las listas se utilizan para almacenar datos del mismo tipo y significado, por lo que el orden puede hacer diferencia. Las tuplas, además de ser inmutables, tienen una semántica mucho más estructural, almacenando datos heterogéneos. Es decir, la posición del dato puede cambiar su significado. También conocemos otro tipo que nos puede ayudar con el uso adecuado de la tupla: la tupla nombrada, generada por la función `namedtuple()`. Con él, podemos darle un nombre a cada elemento de nuestra tupla.

¿Qué tal aprender más sobre **Python** y sus diversos recursos? Entonces, ¡Mira nuestros cursos de **Python para Data Science** aquí en Alura!

Cursos de Data Science

ARTÍCULOS DE TECNOLOGÍA > DATA SCIENCE

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses

¡QUIERO EMPEZAR A ESTUDIAR!

Lógica de Programación | Java

Cursos de Front End

HTML y CSS | JavaScript | React

Cursos de Data Science

Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps

Docker | Linux

Cursos de Innovación y Gestión

Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento